Всероссийская олимпиада школьников по информатике 2025–2026

Региональный этап

Разбор задач

Задача 1. Итоги олимпиады

Заметим, что задачу сформулировать по-другому: для каждой пары индексов (i, j) найдем величину $\max(a_j - a_i, 0)$. Необходимо вычислить сумму этих значений для всех пар (i, j).

Подзадача 1

Заметим, что при $1 \leqslant n \leqslant 1\,000$ можно с помощью вложенного цикла для каждого индекса i от 0 до $n-1$ перебрать все индексы j от 0 до $n-1$ и для всех возможных пар (i, j) суммировать $\max(a_j - a_i, 0)$.

Асимптотика такого решения $O(n^2)$.

Подзадача 2

Если все a_i одинаковые, то нужная сумма равна 0.

Подзадача 3

В этом случае массив из входных данных представляет собой перестановку чисел от 1 до n. Для такого массива школьник с n баллами получил за школьника с одним баллом $n-1$ конфету, за школьника с 2 баллами $n-2$ конфету, и т.д. За школьника с $n-1$ баллом он получит 1 конфету. Таким образом, школьник с n баллами получит $\frac{(n-1)\cdot n}{2}$ конфет — сумму чисел от 1 до $n-1$. Школьник с $n-1$ баллами получит от 1 до $n-2$ конфет за каждого школьника с меньшим количеством баллов, чем у него, то есть $\frac{(n-2)\cdot(n-1)}{2}$ конфет, школьник с $n-2$ баллами $\frac{(n-3)\cdot(n-2)}{2}$ конфет и так далее. Школьник с одним баллом не получит конфет. Таким образом, для получения итогового ответа нужно посчитать сумму $\sum_{i=1}^{n-1} \frac{i\cdot(i+1)}{2}$

Асимптотика такого решения $O(n)$.

Подзадача 4

Если $0 \leqslant a_i \leqslant 1$, то массив состоит только из единиц и нулей. В таком случае конфеты получают только те школьники, которые получили один балл, причем они получают по одной конфете за каждого школьника с нулём баллов. Пусть cnt_0 — количество школьников с нулём баллов, количество школьников с одним баллом тогда $n - cnt_0$. Количество школьников с нулем баллов можно посчитать одним проходом по массиву. Тогда ответом на подзадачу будет $cnt_0 \cdot (n - cnt_0)$.

Асимптотика такого решения $O(n)$.

Подзадача 5

В этой подзадаче массив может быть большим, а значения элементов в нем маленькие. Создадим массив cnt размером 101, где cnt[i] — количество элементов со значением i в массиве a. Такой массив можно создать, например, вот так:

```
cnt = [0] * 101
for x in a:
    cnt[x] += 1
```

После чего для всех i от 1 до 100 переберем j от 0 до $i-1$ прибавим к сумме $(i-j)\cdot cnt[i]\cdot cnt[j]$ — каждый участник с i баллами получит $i-j$ конфет за каждого участника с j баллами. Такой подсчет будет работать за $O(len(cnt)^2)$, но $len(cnt)^2 = 101^2 \leqslant n$.

Таким образом, асимптотика такого решения $O(n)$.

Подзадача 6

Эта подзадача аналогична подзадаче 4. Массив состоит из не более двух различных значений x и y. Пусть $x \leqslant y$. В таком случае конфеты получат только те школьники, которые имеют y баллов, получат они $y - x$ конфет. Пусть cnt_x — количество школьников, у которых x баллов, количество школьников, у которых y баллов, тогда $n - cnt_x$. Количество школьников, у которых x баллов, можно посчитать одним проходом по массиву. Тогда ответом на подзадачу будет $cnt_x \cdot (n - cnt_x) \cdot (y - x)$. Если $x = y$, то ответ получится 0.

Асимптотика такого решения $O(n)$.

Подзадача 7

Заметим, что для каждой пары индексов i и j, где $a_i > a_j$, конфеты получит только школьник i, причем получит он $a_i - a_j$ конфет. Это значение при этом не зависит от положения школьников в исходном массиве, поэтому отсортируем массив a по неубыванию. Рассмотрим школьника с индексом i в отсортированном массиве: он получит $a_i - a_0 + a_i - a_1 + \cdots + a_i - a_{i-1} = i \cdot a_i - \sum_{j=0}^{i-1} a_j$. Посчитать такую сумму для каждого i можно за один проход по массиву: значение $\sum_{j=0}^{i-1} a_j$ — это префиксная сумма элементов на полуинтервале $[0, i)$. Количество элементов, меньших a_i, в отсортированном массиве a будет i.

Итоговое решение выглядит так:

```
n = int(input())
a = list(map(int, input().split()))
a.sort()
res = 0
pref_s = 0
for i in range(n):
#Добавим к ответу значение для индекса $i$
    res += i * a[i] - pref_s
#Увеличим префиксную сумму для следующего элемента
    pref_s += a[i]
print(res)
```

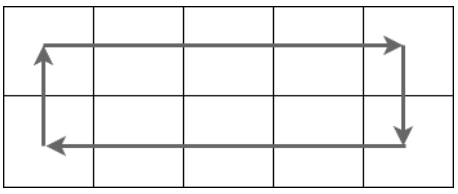
Асимптотика такого решения $O(n)$.

Задача 2. Хромой король

Обозначим за n длину горизонтальной стороны, а m — вертикальной. Также упорядочим клетки, пусть $x_1 \leqslant x_2$; $y_1 \leqslant y_2$.

Рассмотрим несколько случаев:

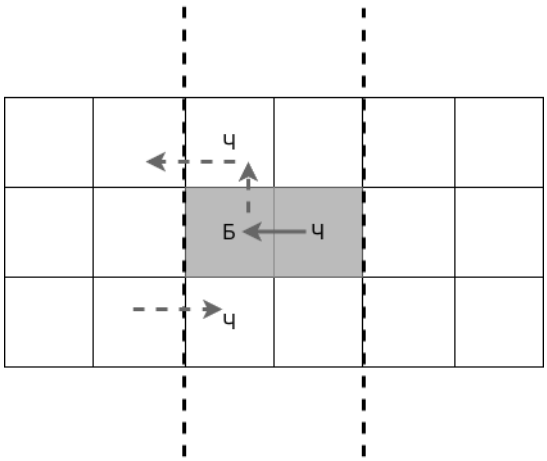
- Если n и m — нечетные, то цикла, проходящего по всем клеткам, не существует.
- Если одна из сторон равняется 2, пусть $n = 2$, то для такой доски существует ровно два обхода, которые различаются только направлением движения.



Поэтому построить обход можно только для отрезков, попадающих в этот путь.

- Если одна из сторон равняется 3, пусть $n = 3$. Пусть также $x_1 = x_2 = 2$.

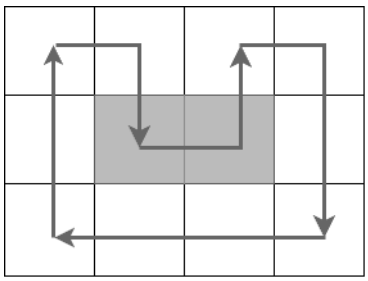
  Покрасим доску в шахматную раскраску, а потом разделим её на 3 части. Заметим, что, проходя по циклу, мы пересечём каждую пунктирную линию ровно два раза. Значит, из белой клетки нашего отрезка мы должны пойти либо вверх, либо вниз.



  Допустим, мы пошли наверх и оказались в черной клетке. Далее из этой клетки мы перейдем пунктирную линию и будем обходить одну из отрезанных частей. Если в этой части четное число клеток, то при выходе из неё мы должны оказаться в клетке противоположного цвета, то есть белого. Однако это не так.

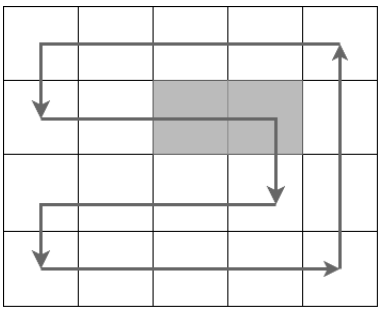
  Значит, если при разбиении доски пунктирными линиями крайние части имеют чётное число клеток(то есть y_1 — нечетное), цикл построить не удастся.

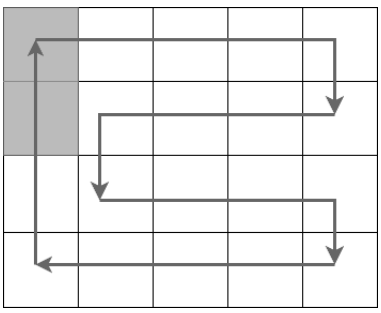
  Если же крайние части состоят из нечетного числа клеток, можно сделать обход «змейкой».



Теперь покажем, как построить путь в остальных случаях. Пусть $n, m \geqslant 4$, n — четное:

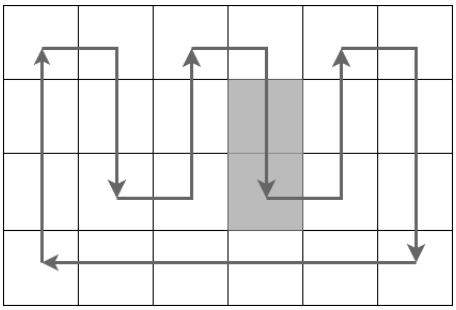
- Если наш отрезок расположен горизонтально, или он лежит в первом или последнем столбце доски, воспользуемся обходом «змейкой».

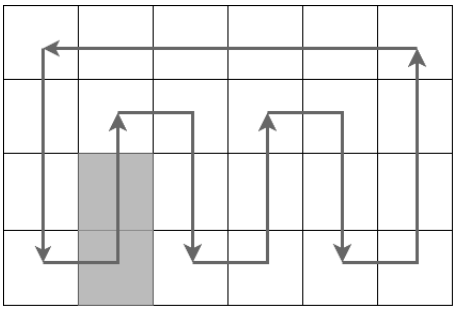




  Заметим, что каждый горизонтальный отрезок попадет хотя бы в одну из этих «змеек».

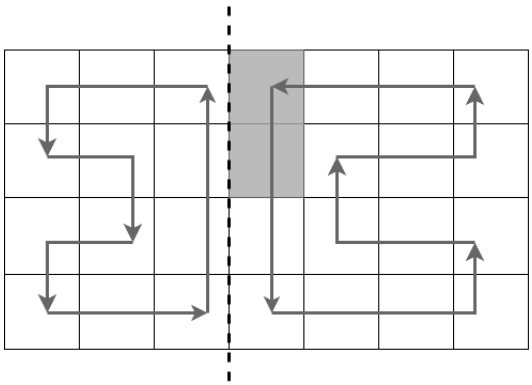
- Если m — четное, отрезок расположен вертикально, можно аналогично воспользоваться обходом «змейкой».



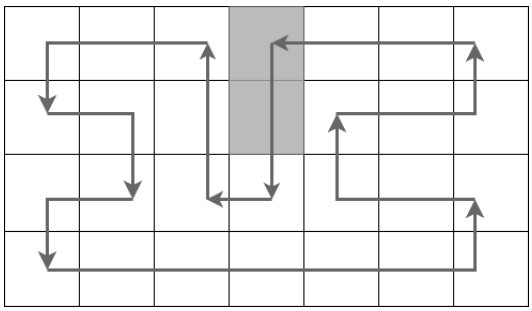


- Остался случай, когда m —нечетное, отрезок расположен вертикально, отрезок не лежит в первом и последнем столбце.

  Разобьем нашу доску на 2 части так, чтобы каждая состояла хотя бы из двух столбцов и в одной из них наш отрезок лежал с краю. Мы всегда сможем так сделать, поскольку $m \geqslant 4$ и отрезок лежит не с краю. Заметим, что каждую из этих частей мы можем обойти «змейкой».



Объединим эти два обхода, и получим цикл для всей доски.



Задача 3. Расстановки фишек

Подзадача 1. $n \leqslant 10, m \leqslant 4$. В этой подзадаче достаточно полного перебора за $2^{m \times m}$.

Подзадача 2. $n = 1, m \leqslant 1000$. В этой подзадаче достаточно заметить, что среди $r_1 \cdot c_1$ клеток может быть выбрано не более одной, то есть всего на этом подпрямоугольнике $(r_1+1)(c_1+1)$ способов. А оставшиеся $m^2 - r_1 \cdot c_1$ могут как входить, так и нет. Поэтому, ответ равен $(r_1+1)(c_1+1)2^{m^2-r_1c_1}$

Подзадача 3. $n \leqslant 10, m \leqslant 1000$**.** Для решения этой подзадачи можно для каждого подмновества ограничений посчитать число способов, которые покравают это подмножество. Дальше с помощью динамического программирования на подмножествах посчитать итоговый ответ. Здесь можно было реализовать решение за $\mathcal{O}(4^n)$.

Подзадача 4. $n \leqslant 15, m \leqslant 10^9$**.** Здесь то же самое, как в предыдущей подзадаче, но необходимо быть более эффективным для подсчета количества точек для фиксированного подмножества ограничений. Например, можно было воспользоваться методом включений исключений за $\mathcal{O}(3^n)$.

Подзадачи 5–10 Для решения остальных подзадач сначала упростим набор ограничений. Если существуют два ограничения (r_i, c_i) и (r_j, c_j) такие, что $r_i \leqslant r_j$ и $c_i \leqslant c_j$, то первое из них избыточно, поскольку соответствующий прямоугольник полностью содержится во втором. Отсортируем все ограничения по возрастанию r, а при равенстве — по возрастанию c, и пройдём по ним монотонным стеком, поддерживая строго убывающую последовательность по c. В результате останется набор ограничений, для которого выполняется

$$r_1 \leqslant r_2 \leqslant \ldots \leqslant r_n, \qquad c_1 \geqslant c_2 \geqslant \ldots \geqslant c_n.$$

После этого каждую клетку доски можно охарактеризовать следующим образом: либо она не покрыта ни одним ограничением, либо она покрыта непрерывным отрезком ограничений $[i, j]$. Если выбрать две клетки, чьи отрезки ограничений пересекаются, то найдётся ограничение, покрывающее обе клетки, а значит в соответствующем прямоугольнике окажутся две фишки, что запрещено. Поэтому допустимы только такие наборы клеток, у которых отрезки ограничений не пересекаются.

Клетки, не покрытые ни одним ограничением, полностью независимы от остальных: в каждой из них фишку можно либо поставить, либо нет. Если таких клеток K, то они дают множитель 2^K к ответу.

Для остальных клеток будем считать, что каждая из них соответствует отрезку $[i, j]$. Обозначим через $cnt[i][j]$ количество клеток, которые покрыты ровно ограничениями с i по j. Перебирать клетки напрямую возможно для прохождения подзадач 5–7 (отрезок для клетки можно найти бинарным поиском), для остальных можно заметить, что $cnt[i][j]$ легко считается по формуле включения–исключения. Количество клеток, лежащих в прямоугольнике $[1..r] \times [1..c]$, равно $r \cdot c$, поэтому при $1 \leqslant i \leqslant j \leqslant n$

$$cnt[i][j] = (r_j c_i) - (r_{j-1} c_i) - (r_j c_{i+1}) + (r_{j-1} c_{i+1}),$$

где для удобства считаем $r_0 = 0$ и $c_{n+1} = 0$.

Теперь введём динамическое программирование. Пусть $dp[x]$ — количество способов корректно выбрать клетки, учитывая только первые x ограничений. Изначально $dp[0] = 1$. При переходе к следующему ограничению либо мы не выбираем ни одной клетки, начинающейся в этом месте, либо выбираем одну клетку с отрезком $[i, j]$. Получаем формулу

$$dp[j+1] = dp[j] + \sum_{i=1}^{j} dp[i-1] \cdot cnt[i][j].$$

Подставляя выражение для $cnt[i][j]$ и раскрывая скобки, получаем

$$dp[j+1] = dp[j] + (r_j - r_{j-1}) \sum_{i=1}^{j} dp[i-1]\,(c_i - c_{i+1}).$$

Если поддерживать сумму

$$S_j = \sum_{i=1}^{j} dp[i-1]\,(c_i - c_{i+1}),$$

то переход считается за $O(1)$, и вся динамика работает за линейное время.

Итоговый ответ равен

$$dp[n] \cdot 2^K \bmod (10^9 + 7),$$

где K — количество клеток, не покрытых ни одним ограничением. Общая асимптотика решения составляет $\mathcal{O}(n \log n)$ из-за сортировки ограничений.

Для решения только подзадач 6–8, достаточно было реализовать динамику за $\mathcal{O}(n^2)$.

Также для подзадачи 9 можно было реализовать решения, которые работают за более долгое время, например $\mathcal{O}(n + m)$

Задача 4. Прыжки по вершинам

Для начала сформулируем условие более формально: для каждого отрезка зубцов необходимо было определить количество рёбер в строго выпуклом вверх маршруте из начала отрезка в конец, построенном на точках отрезка, таком, что все точки отрезка находятся под ним. Видно, что такой маршрут всегда существует и единственен.

Подзадача 1. $n, q \leqslant 300$**.** Здесь можно было для каждого отрезка запроса восстанавливать последовательность прыжков честно, каждый раз выбирая подходящий прыжок. Заметим, что подходящий прыжок — это прыжок с наибольшим возможным углом (и из всех таких — с наибольшей длиной). Получаем решение, работающее за $O(qn^2)$.

Подзадача 2. $n, q \leqslant 5000$**.** Здесь можно было заметить, что ответ на каждый вопрос можно вычислять за линию от длины отрезка при помощи стека. Делать это будем следующим образом: заведём стек, в котором будем хранить предполагаемые зубцы маршрута Пети. Если при добавлении очередного зубца в стек оказывается, что последний прыжок Пети будет иметь не меньший угол, чем предпоследний, выкинем последний зубец из стека и выполним проверку ещё раз. Нетрудно заметить, что такой алгоритм строит искомый выпуклый маршрут. Получаем решение, работающее за $O(qn)$.

Подзадача 3. $h_i \leqslant 10$ Обозначим за h максимальную высоту зубца. Пусть мы хотим ответить на запрос $[l, r]$. Найдём вторую после l точку маршрута. Заметим, что из всех точек одинаковой высоты всегда оптимально будет выбрать самую близкую к l или самую дальнюю от неё. Давайте среди оптимальных точек каждой высоты найдём вторую точку маршрута (для каждой из n точек самую близкую точку данной высоты слева и справа можно предподсчитать за $O(nh)$ суммарно). При этом заметим, что точек в маршруте будет $O(h)$, значит, суммарное время работы будет $O(nh + qh^2)$.

Для полного решения нам понадобится следующая идея: пусть мы знаем зубцы, по которым будет прыгать Петя на отрезке от зубца i до зубца k, а также от зубца k до зубца j. Научимся быстро находить набор зубцов на пути от i до j. Легко видеть, что искомый набор состоит из префикса зубцов на пути от i до k и суффикса зубцов на пути от k до j. Давайте будем перебирать длину искомого префикса по возрастанию длины, а для каждого префикса — длину искомого суффикса (тоже по возрастанию длины) и проверять, что многоугольник с заданными префиксом и суффиксом будет выпуклым. Получаем, что «слияние» двух оболочек будет работать за $O(n^2)$. А если заметить, что оба перебора можно заменить на бинарный поиск — за $O(\log^2 n)$.

Подзадача 4. Один конец отрезка находится в левой половине массива, другой — в правой. В этой подзадаче для каждого отрезка нам, по сути, нужно слить его «левую» и «правую» части (относительно середины массива) с помощью описанной выше идеи. Но для этого нам нужно уметь эффективно делать бинпоиск по этим частям. Воспользуемся следующей идеей. Начнём от середины массива и будем идти вправо, поддерживая стек пути, как в подзадаче 2. Тогда на i-м шаге мы будем иметь в стеке путь Пети до i-го с середины элемента. Давайте назовём предком i-го элемента предшествующий ему элемент в стеке в этот момент (т.е. предпоследний зубец на пути от середины до i). В результате мы получим дерево, в котором путь от корня до i-го элемента совпадает с путём Пети от середины до него. Насчитав на этом дереве двоичные подъёмы (и сделав такое же

дерево для левой половины), мы сможем делать нужный нам бинпоиск. Получаем $O(n \log n)$ на предподсчёт и $O(q \log^2 n)$ на все вопросы.

Подзадача 5. $n, q \leqslant 5 \cdot 10^4$**.** В эту подгруппу можно сдавать неэффективные версии полного решения (в том числе неэффективные вариации бинпоиска по путям, без деревьев).

Подзадача 6. Полное решение. Применим метод разделяй-и-властвуй. Т.е. запустимся от всего массива, рекурсивно ответим на запросы, целиком лежащие в левой и правой половинах, а потом обработаем запросы, пересекающие середину, так же, как и в предыдущей подзадаче. Решение работает за $O(n \log^2 n + q \log^2 n)$.

Задача 5. Покраска бруска

Заметим, что формула зависит от количества единичных сторон у бруска.

1. Если все стороны равны единице, то у нас ровно один единичный куб, у которого будет 6 покрашенных сторон.

2. Если две стороны равны единице (пусть это будут b и c), то мы имеем полоску из единичных кубиков. В ней два крайних кубика будут иметь 5 покрашенных сторон, а остальные $a - 2$ — 4 покрашенные стороны.

3. Если ровно одна сторона имеет размер 1 (пусть $c = 1; a, b \neq 1$), мы получаем . Тогда четыре угловых кубика у бруска будут иметь 4 покрашенных стороны.

   Кубики, лежащие на ребре (их будет $2(a-2)+2(b-2)$), будут иметь по 3 покрашенных стороны.

   У остальных $(a - 2) \cdot (b - 2)$ кубиков будет ровно 2 покрашенных стороны.

4. При отсутствии единичных сторон брусок будет распилен на 4 вида кубиков.

   Восемь угловых кубиков будут иметь по 3 покрашенных стороны.

   Остальные $4(a-2)+4(b-2)+4(c-2)$ кубика на ребрах будут иметь 2 покрашенных стороны.

   У кубиков на грани (всего их будет $(a-2)(b-2)+(b-2)(c-2)+(c-2)(a-2)$) будет одна покрашенная сторона.

   Оставшиеся $(a-2)(b-2)(c-2)$ внутренних кубиков не будет иметь покрашенных сторон.

Задача 6. Битовая магия

Условие $x \,\&\, b = b$ означает, что во всех битах, где у числа b стоит единица, у числа x тоже обязана стоять единица. В битах, где у b стоит ноль, число x может содержать как 0, так и 1.

Для подсчёта количества подходящих чисел на отрезке $[l, r]$ будем использовать стандартную формулу:

$$ans(l, r) = count(r) - count(l - 1),$$

где $count(N)$ — количество чисел x, таких что $0 \leqslant x \leqslant N$ и $x \,\&\, b = b$.

Подзадачи 1–4

В подзадачах 1–4 ограничения достаточно маленькие $r, b < 16^7$, поэтому все решения можно реализовать *в лоб*.

Для каждой подзадачи перебираем все числа x от l до r и проверяем условие $(x \,\&\, b) = b$.

Такой перебор работает за $O(r - l)$ и укладывается во все ограничения этих подзадач. Никаких оптимизаций или динамики здесь не требуется.

Подзадачи 5–6

В подзадачах 5–6 числа уже достаточно большие, но всё ещё помещаются в стандартный тип `long long`.

Здесь используется уже *полное решение*, но без работы с длинной арифметикой:

- считаем функцию $count(N)$ для одного числа N;
- используем побитовую динамику по двоичному представлению числа;
- идём по битам слева направо и поддерживаем флаг, показывающий, что текущее число уже стало меньше N.

Для каждого бита:

- если бит $b_i = 1$, то в x можно поставить только 1;
- если $b_i = 0$, то можно поставить 0 или 1 (с учётом ограничения $x \leqslant N$).

Так как количество битов ограничено (не более 60), решение работает за $O(n)$ и полностью проходит подзадачи 5–6.

Подзадачи 7–8

В этих подзадачах числа уже не помещаются в стандартные типы и имеют длину до 16^{1000}, однако решение всё ещё основано на динамике.

Для вычисления $count(N)$:

- переводим число N и маску b в двоичное представление;
- перебираем позицию i, в которой число x впервые становится строго меньше N;
- проверяем, что префикс не противоречит маске b;
- считаем количество возможных продолжений на суффиксе.

Подсчёт количества допустимых вариантов на суффиксе выполняется прямым перебором битов, поэтому для каждой позиции i требуется $O(n)$ операций.

Общая сложность такого решения — $O(n^2)$, чего достаточно для подзадач 7–8, но недостаточно для максимальных ограничений.

Подзадача 10

Подзадача 10 по сути аналогична предыдущим, но с дополнительным упрощением: $b = 0$.

Условие $x \,\&\, 0 = 0$ выполняется для любого числа $x \geqslant 0$, поэтому подходят все числа на отрезке $[l, r]$.

Ответ равен:

$(r - l + 1) \bmod (10^9 + 7)$,а все вычисления выполняются с длинными числами, заданными в шестнадцатеричной системе.

Подзадачи 9 и 11

В подзадачах 9 и 11 длина чисел достигает 50 000 шестнадцатеричных символов, что соответствует примерно 200 000 битам. Решение с квадратичной сложностью здесь уже не проходит.

Основная оптимизация заключается в ускорении подсчёта количества вариантов на суффиксе.

- Заранее вычисляется массив $suffix\_zeros[i]$, равный количеству нулевых битов в маске b на позициях $j \geqslant i$.
- Также предварительно считаются степени двойки $2^k \bmod (10^9 + 7)$.

Теперь при рассмотрении позиции i, в которой x становится меньше N, количество допустимых продолжений на суффиксе считается за $O(1)$ как $2^{suffix\_zeros[i+1]}$.

Это позволяет убрать внутренний цикл и свести вычисление $count(N)$ к линейному времени.

Полное решение

Итоговый алгоритм:

1. Перевести числа l, r и b из шестнадцатеричной записи в двоичную.
2. Предварительно посчитать количество нулей в маске b на каждом суффиксе и степени двойки.
3. Реализовать линейный подсчёт $count(N)$.
4. Получить ответ как $(count(r) - count(l) + check(l) + (10^9 + 7)) \bmod (10^9 + 7)$, где $check(l)$ проверяет, удовлетворяет ли число l условию $l \,\&\, b = b$.

Время работы итогового решения — $O(n)$, что позволяет пройти подзадачи 9 и 11 и получить полный балл.

Задача 7. Скользящие окна

Подзадачи 1–3

Будем проходиться по всем окнам в запросах явно. Есть несколько вариантов искать на них минимумы:

- поиск минимума на отрезке за его длину даёт решение за $O(n^2 q)$ (подзадача 1)
- дерево отрезков на минимум — $O(nq \log n)$ (подзадача 2, подзадача 3 с эффективной реализацией ДО)
- явный предподсчёт всех минимумов — $O(nq + n^2)$ (подзадача 2)
- разреженные таблицы (sparse table) — $O(nq + n \log n)$ (подзадача 3)

Подзадачи 3–4

Сгруппируем запросы по k и будем решать задачу для каждого k отдельно. Для фиксированного k мы можем найти минимумы на скользящих окнах за $O(n)$. После этого мы можем отвечать на запросы либо за $O(n)$ ($O(n^2 + nq)$, подзадача 3), либо за $O(1)$ с префиксными суммами ($O(n^2 + q)$, подзадача 4).

Подзадача 5

Аналогично решению для подзадачи 4, найдём минимумы на всех скользящих окнах длины k за $O(n)$, после чего посчитаем на них префиксные суммы, чтобы отвечать на запросы за $O(1)$. Это даёт решение за $O(n + q)$.

Подзадачи 6–7

Если все элементы массива равны 1 или 2, то минимум равен 2 только на отрезках, полностью состоящих из 2. Для каждой позиции i найдём максимальное k, такое что отрезок $[i; i + k - 1]$ полностью состоит из 2.

Отсортируем запросы по возрастанию k. Для каждой позиции i будем поддерживать значение минимума на отрезке $[i; i + k - 1]$. Действительно, с возрастанием k минимум на таком отрезке не

более чем единожды меняется с 2 на 1, и для каждого k мы можем сохранить список позиций, для которых это произойдёт в этот момент.

Тогда ответ на очередной запрос равен сумме на некотором отрезке таких минимумов. Чтобы эффективно найти её, будем поддерживать минимумы в дереве отрезков или дереве Фенвика. Это даёт нам решение за $O((n+q)\log n)$ (подзадача 6).

Аналогично, если элементы в массиве не превышают A, то с возрастанием k минимум на отрезке $[i; i+k-1]$ изменится не более $A-1$ раз. Найти k, при которых это происходит, можно, пройдясь по массиву справа налево, поддерживая стек рекордных минимумов. Это даёт нам решение за $O((n+q)A\log n)$ (подзадача 7). Поскольку $A \leqslant n$, аккуратная реализация этого решения также проходит подзадачи 1–2.

Подзадача 8

В решении прошлой подзадачи мы поддерживали значения минимумов на скользящих окнах длины k. Заметим, что каждый элемент a_i является минимумом на некотором подотрезке окон (возможно, пустом). Для простоты скажем, что, если $a_i = a_j$, то мы считаем меньшим элемент с меньшим индексом.

Пусть L и R — индексы ближайшего к i элемента, меньшего a_i, слева и справа соответственно ($L = 0$ или $R = n+1$, если такого не существует). Тогда a_i будет минимумом на отрезке $[l; r]$ тогда и только тогда, когда $L < l \leqslant i \leqslant r < R$.

Для каждого k найдём количество окон длины k, на которых a_i является минимумом:

- k, если $k \leqslant \min(R-i, i-L)$
- $\min(R-i, i-L)$, если $\min(R-i, i-L) \leqslant k \leqslant \max(R-i, i-L)$
- $R-L-k$, если $\max(R-i, i-L) \leqslant k \leqslant R-L$
- 0, если $k \geqslant R-L$

Таким образом, вклад i-го элемента в сумму минимумов на всех скользящих окнах длины k (a_i, умноженное на количество) — кусочно-линейная функция. Чтобы найти L и R для всех i, можно воспользоваться известным линейным алгоритмом со стеком рекордов.

В подзадаче 8 нужно найти значение в фиксированном k суммы этих кусочно-линейных функций по всем i. Сумма кусочно-линейных функций — это кусочно-линейная функция, которую можно найти, если просуммировать по всем i изменения наклона в каждой точке k.

Подзадача 9 (полное решение)

В полном решении нужно найти сумму минимумов на некотором подотрезке окон, то есть мы должны просуммировать не все кусочно-линейные функции, а только некоторый их подотрезок, и дополнительно учесть значения на краях отрезка.

Рассмотрим запрос (l, r, k). Пусть a_i — минимум на отрезке $[l; l+k-1]$, a_j — минимум на отрезке $[r-k+1; r]$. Если $i = j$, то a_i будет минимумом на всех рассматриваемых окнах. Иначе a_i будет минимумом на $\min(i-l+1, R_i-l-k+1)$ окнах, a_j будет минимумом на $\min(r-j-1, r-L_j-k-1)$ окнах.

Теперь осталось просуммировать кусочно-линейные функции для позиций с $i+1$ по $j-1$. В каждый момент времени мы можем хранить их как линейные функции, изменяя их в точках $\min(R_i-i, i-L_i)$, $\max(R_i-i, i-L_i)$, R_i-L_i. Тогда, построив дерево отрезков или дерево Фенвика на линейных функциях, мы можем найти их сумму на отрезке. Подставив текущее значение k в получившуюся линейную функцию, мы получим вклад элементов $a_{i+1}, \ldots, a_{j-1}$.

Таким образом, мы получаем полное решение за $O((n+q)\log n)$.

Задача 8. XOR Раскраска

Подзадача 1. $n \leqslant 2$ Здесь достаточно простого разбора случаев.

Подзадача 2. $n \leqslant 5$ Здесь достаточно полного перебора всех различных раскрасок.

Переформулируем задачу как задачу раскраски графа: вершины соответствуют элементам массива A, и между вершинами x и y проводится ребро, если существует такой j, что $(a_x \oplus b_j) \leqslant x$ и $(a_y \oplus b_j) \leqslant x$. Требуется найти хроматическое число этого графа.

Подзадача 3. $n \leqslant 15$ Эта задача может быть решена с помощью поиска хроматического числа с помощью динамики по подмножествам.

Подзадача 4. $n \leqslant 100$ Для решения этой подзадачи можно попробовать написать жадный поиск раскраски. В произвольном порядке $1, 2 \ldots, n$ будем красить элемент i в минимальный цвет, который отличается от всех элементов $< i$ соединенных с i ребром. Доказательство исходит из полного решения.

Подзадача 5. $n \leqslant 2\,000$ Здесь достаточно оптимизировать решение подзадачи 4 с помощью битового сжатия.

Подзадача 6. $n \leqslant 5\,000$ Здесь достаточно во время раскраски поддерживать объединение всех $S(i)$ для каждого цвета.

Будем решать задачу рекурсивно, двигаясь от старших битов к младшим. Рассмотрим функцию решения для тройки (i, A, B), где учитываются только биты с номерами от i до 0. Если на каком-то шаге массив A или массив B оказывается пустым, то ответ равен 0.

Для фиксированного бита i разобьём массивы по значению этого бита: $X[0]$ и $X[1]$ — элементы массива A, у которых i-й бит равен 0 и 1 соответственно. Аналогично определим $Y[0]$ и $Y[1]$ для массива B.

Если i-й бит числа x равен 0, то для любых $a \in X[0]$ и $b \in Y[1]$ значение $(a \oplus b)$ уже превышает x по i-му биту, и такие пары нас не интересуют. Поэтому задача распадается на две независимые: для $(i-1, X[0], Y[0])$ и $(i-1, X[1], Y[1])$. В этом случае минимальное число цветов равно

$$\max\big(\text{ans}(i-1, X[0], Y[0]),\ \text{ans}(i-1, X[1], Y[1])\big).$$

Меньшее значение невозможно, а равенство достигается за счёт объединения раскрасок.

Теперь рассмотрим случай, когда i-й бит числа x равен 1. Если $Y[0]$ пусто, то для любого элемента из $X[1]$ множество $S(i)$ содержит все элементы массива B. Следовательно, никакие два элемента из $X[1]$ не могут иметь одинаковый цвет. Тогда ответ равен

$$|X[1]| + \text{ans}(i-1, X[0], Y[1]).$$

Аналогично рассматривается ситуация, когда $Y[1]$ пусто.

Остаётся случай, когда оба множества $Y[0]$ и $Y[1]$ непусты. В этом случае любые два элемента из $X[0]$ конфликтуют между собой, и то же верно для любых двух элементов из $X[1]$. Значит, допустимые цветовые группы могут быть либо одиночными элементами, либо парами, состоящими из одного элемента из $X[0]$ и одного элемента из $X[1]$.

Однако не всякую такую пару можно образовать. Если элемент из $X[0]$ содержит в своём множестве $S(i)$ хотя бы один элемент из $Y[1]$, то он конфликтует со всеми элементами из $X[1]$ и не может входить в пару. Аналогично для элементов из $X[1]$, которые содержат в $S(i)$ элементы из $Y[0]$.

Пусть p_0 — количество элементов из $X[0]$, которые не содержат в $S(i)$ ни одного элемента из $Y[1]$, а p_1 — количество элементов из $X[1]$, которые не содержат в $S(i)$ ни одного элемента из $Y[0]$. Эти величины можно посчитать, спускаясь по битовому бору.

Тогда максимальное число допустимых пар равно $\min(p_0, p_1)$, и ответ равен

$$|X[0]| + |X[1]| - \min(p_0, p_1).$$

Таким образом, рекурсивно обрабатывая биты от старших к младшим и разбивая массивы по значениям текущего бита, можно найти минимальное число цветов, необходимое для корректной раскраски массива A.

Подзадачи 7–8 Эти подзадачи можно пройти с помощью аккуратного разбора случаев или поиском p_0, p_1 за $\mathcal{O}(nm)$

Подзадачи 9–10 Эти подзадачи можно пройти с помощью аккуратного разбора случаев или перебора который учитывает некоторые идеи из полного решения. Также можно было справиться с помощью полного решения на поиском p_0, p_1 за $\mathcal{O}(nt)$, где t – количество различных элементов в B.